SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Stručni studij elektrotehnike, smjer Informatika

WEB APLIKACIJA ZA PRAĆENJE OSOBNE POTROŠNJE

Završni rad

Hrvoje Vučetić

Osijek, 2019

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1S: Obrazac za imenovanje Povjerenstva za obranu završnog rada na preddiplomskom stručnom studiju

Osijek, 18.09.2019.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu završnog rada na preddiplomskom stručnom studiju

Ime i prezime studenta:	Hrvoje Vučetić
Studij, smjer:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI 4418, 19.10.2018.
OIB studenta:	33419240131
Mentor:	Prof. dr. sc. Dominika Crnjac Milić
Sumentor:	Izv. prof. dr. sc. Krešimir Nenadić
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Alfonzo Baumgartner
Član Povjerenstva:	Doc.dr.sc. Tomislav Rudec
Naslov završnog rada:	Web aplikacija za praćenje osobne potrošnje
Znanstvena grana rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak završnog rada	
Prijedlog ocjene pismenog dijela ispita (završnog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	18.09.2019.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 03.10.2019.

Ime i prezime studenta:	Hrvoje Vučetić
Studij:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI 4418, 19.10.2018.
Ephorus podudaranje [%]:	8%

Ovom izjavom izjavljujem da je rad pod nazivom**: Web aplikacija za praćenje osobne potrošnje**

izrađen pod vodstvom mentora Prof. dr. sc. Dominika Crnjac Milić

i sumentora Izv. prof. dr. sc. Krešimir Nenadić

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. UVOD

Sve užurbaniji i dinamičniji životni ritam uzrokuje povećanje broja novčanih transakcija što može znatno otežati praćenje svakodnevnih troškova te samim time organizaciju i raspodjelu raspoloživim novčanim sredstvima. Različiti oblici potrošnje novca koji su nam dostupni u današnje vrijeme, kao što su online kupovina, više vrsta kartičnog, te naravno gotovinsko plaćanje, osobu lako može dovesti do previda ukupno ostvarene potrošnje u određenom vremenskom periodu.

Aplikacija za praćenje osobne potrošnje pruža priliku za točnu i preciznu kontrolu ostvarene potrošnje odnosno odljeva sredstava iz osobnog budžeta. Isto tako omogućuje i praćenje priljeva sredstava odnosno ostvarenih novčanih prihoda. Redovito praćenje osobnih financija može potaknuti na razvijanje boljih potrošačkih navika te osobu potaknuti na štednju što dugoročno može rezultirati financijskom stabilnošću i sigurnošću pojedinca.

Nakon uvoda, drugim poglavljem dan je osvrt na primjenjivost i ekonomsku korisnost aplikacije te je pruženo teorijsko pojašnjenje tehnologija korištenih prilikom izrade aplikacije. Nastavno na teorijski osvrt, kojim su pobliže opisane korištene tehnologije te njihova uloga u izradi aplikacije, opisan je nastanak same aplikacije uz predočenje koda na kojem se ista zasniva te je prikazan način korištenja aplikacije uz pojašnjenje glavnih komponenata.

1.1. Zadatak završnog rada

Zadatak završnog rada je kreirati web aplikaciju koja korisniku putem privatnog korisničkog računa omogućuje unos prihoda na mjesečnoj te troškova na dnevnoj bazi. Korisnik ima mogućnost sortirati i filtrirati unesene troškove po iznosu, datumu unosa te nazivu samog troška. Također, korisniku je omogućeno naknadno uređivanje postojećih troškova i prihoda te potpuno uklanjanje istih.

2. PRIMJENA APLIKACIJE I TEORIJSKO POJAŠNJENJE TEHNOLOGIJA

Ovim poglavljem pružen je kratak osvrt na primjenu i korisnost Web aplikacije za praćenje osobne potrošnje te glavne prednosti koje korisnik ostvaruje korištenjem iste.

Nastavno na pojašnjenje primjene i korisnosti same aplikacije, detaljnije su pojašnjene tehnologije koje su korištene prilikom izrade aplikacije za praćenje osobnih troškova. Cilj poglavlja je ukazati na prednosti korištenja aplikacije kao i pojasniti ključne značajke korištenih tehnologija u svrhu boljeg shvaćanja praktične primjene istih te uloge svake pojedine tehnologije u izradi aplikacije.

2.1. Primjena i prednosti korištenja aplikacije

Sprječavanje neželjene potrošnje odnosno prekomjerne potrošnje postaje sve teža zadaća, čemu svakako doprinosi ubrzani i dinamični životni stil današnjeg vremena. Svakodnevno se susrećemo s mnoštvom troškova manjih iznosa čiji ukupni iznos, dugoročno gledajući, može postati značajan problem ukoliko isti ne pratimo te ne kontroliramo. Sve brži i užurbaniji način života tjera nas na mnoštvo novčanih izdataka na dnevnoj bazi. Kako je većina svakodnevnih troškova relativno niskog iznosa, iste često možemo previdjeti te im ne pridajemo jednaku pažnju kao većim troškovima koji zauzimaju značajniji iznos naših mjesečnih primanja.

Međutim, upravo prethodno spomenuti troškovi mogu prouzročiti "rupu" u našem osobnom proračunu odnosno problem u organizaciji raspoloživim novčanim sredstvima. Ukoliko na dnevnoj bazi ne pratimo potrošnju te ne vodimo računa o tome je li ista prekomjerna ili nije, a ovisno o novčanim prihodima s kojima raspolažemo, možemo se dovesti do neželjenih problema te se naći u nepovoljnoj situaciji. Dugoročno gledajući manjak svijesti o ostvarenim troškovima te eventualnoj prekomjernoj potrošnji na dnevnoj bazi može dovesti do formiranja loših potrošačkih navika koje mogu uzrokovati značajne probleme.

Izrađena web aplikacija za praćenje osobne potrošnje pruža rješenje spomenutog problema te na vrlo jednostavan način omogućava praćenje svakodnevnih troškova u svrhu izbjegavanja prekomjerne potrošnje te formiranja loših potrošačkih navika. Aplikacija za praćenje osobne potrošnje omogućuje jednostavan unos svakodnevnih troškova te filtriranje istih po datumu unosa, iznosu ili unesenom nazivu troška te ujedno omogućava uvid u preostala raspoloživa novčana sredstva. Unesene troškove kao i mjesečni prihod, moguće je naknadno uređivati ili čak u potpunosti ukloniti ovisno o potrebama korisnika.

Ideja aplikacije omogućiti je krajnjem korisniku lakše praćenje osobnih potrošačkih navika te

ukazati na eventualne potrebe za korigiranjem istih, a jednostavnost korištenja aplikacije čini je iznimno pogodnom za današnji stil života budući da je aplikacija pregledna i jednostavna za korištenje.

2.2. HTML prezentacijski jezik

HTML (engl. *Hypertext Markup Language*) prezentacijski je jezik, a koristi se za oblikovanje sadržaja web stranice. Bitno je napomenuti kako se ne radi o programskom jeziku jer pomoću HTML-a ne možemo izvršiti niti najjednostavnije logičke operacije kao na primjer zbrajanje ili oduzimanje. HTML tags prvi je javno dostupan opis HTML-a koji se prvi puta spomenuo 1991. godine, a isti se sastojao od samo 20 elemenata početnog i relativno jednostavnog dizajna HTML-a. Prva službena verzija HTML jezika objavljena je 1993. godine dok se trenutno koristi HTML5 standard.

Temeljna zadaća HTML jezika jest uputiti web preglednik kako prikazati hipertext dokument. Pri tome se nastoji da taj dokument izgleda jednako bez obzira o kojemu je web pregledniku, računalu i operacijskom sustavu riječ. [1]

Svaki HTML dokument sastoji se od HTML elemenata koji su osnovni građevni blokovi dokumenta dok se svaki HTML element sastoji od para HTML oznaki (engl. *tags*). Slikom 2.1. prikazan je HTML element, a nastavno na sliku pruženo je detaljnije pojašnjenje iste.

Start tag	Element content	End tag
<h1>	My First Heading	</h1>
	My first paragraph.	

Sl. 2.1. Glavni dijelovi HTML elementa [2]

Kao što je vidljivo na gore prikazanoj slici 2.1., HTML element sastoji se od otvarajuće i zatvarajuće oznake, naziva elementa te sadržaja elementa. Otvarajuća oznaka (engl. *Start tag*) sastoji se od naziva elementa upisanog između otvarajuće "<" i zatvarajuće ">" zagrade dok se zatvarajuća oznaka (engl. *Closing tag*) od otvarajuće oznake razlikuje dodavanjem znaka "/" prije naziva elementa. Između otvarajuće i zatvarajuće oznake upisuje se sam sadržaj HTML elementa.

Također, bitno je napomenuti kako HTML elementi mogu sadržavati i atribute. Atributi, kao što su recimo *class* ili *id*, dodaju se u otvarajuće oznake HTML elemenata te služe kako bi se dobilo

više informacija o elementu bez da iste budu vidljive u sadržaju elementa.

2.3. CSS stilski jezik

CSS (engl. *Cascading Style Sheets*) stilski je jezik namjenjen opisivanju odnosno prezentiranju dokumenata napisanih HTML jezikom. Drugim riječima, primjenom CSS-a određuje se izgled i pozicija HTML elemenata. CSS je razvijen i održavan od strane World Wide Web Consortium odnosno W3C-a.

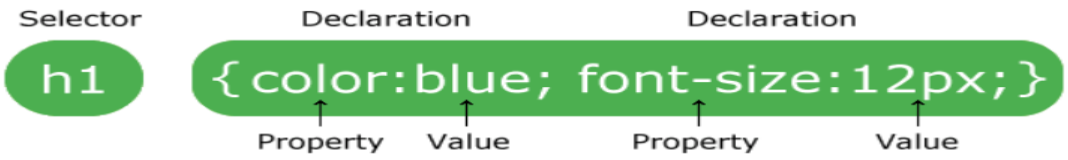
Zbog sve veće složenosti i primjene različitih stilova isti se najčešće pišu u posebne datoteke, tako zvane "*Style sheets*". Spomenute datoteke s pripadajućim stilovima koje se želi primijeniti na pojedine HTML elemente zatim je potrebno "*linkati*" odnosno uvesti u HTML dokument u kojem iste želimo primijeniti. Slika 2.2. prikazuje prethodno spomenut proces dodavanja CSS datoteke u HTML datoteku.

```
<link rel="stylesheet" type="text/css" href="/styles.css"/>
```

Sl. 2.2. Dodavanje CSS datoteke u HTML dokument

Style sheet u CSS-u sastoji se od više pravila dok se svako pravilo sastoji se od dva dijela, tj. selectora i deklaracijskog bloka.

Selektorom odabiremo na koji dio HTML-a želimo primijeniti određeni stil dok se u deklaracijski blok upisuju željena svojstva i vrijednosti odabranih svojstva. Navedeno je prikazano slikom 2.3. u nastavku.



Sl. 2.3. Sintaksa CSS pravila [3]

2.4. JavaScript programski jezik

JavaScript (skraćeno JS) skriptni je programski jezik koji uz gore navedene HTML i CSS čini jednu od temeljnih tehnologija za izradu web stranica i web aplikacija. Primjena JavaScripta omogućuje interaktivnost web stranica, a JS je i bitan dio web aplikacija. JavaScript predstavlja glavnu tehnologiju odnosno alat za izradu aplikacije za praćenje osobnih troškova.

Iako postoje određene sličnosti između JavaScritpa i Jave, uključujući i samo ime te sintaksu bitno je naglasiti kako se radi o dva različita programska jezika koja se uvelike razlikuju. JavaScript iskoristila je popularnost Java programskog jezika te je sintaksa koja se koristi u JS-u namjerno kreirana kako bi što više nalikovala sintaksi Jave i C++ jezika, a u svrhu lakšeg usvajanja i savladavanja JS programskog jezika.

JavaScript kod izvršava se na korisničkoj strani (engl. *client-side*) web stranice ili aplikacije te je najpopularniji skriptni jezik kojeg podržavaju gotovo svi web preglednici. JavaScript podržava programiranje temeljeno na događajima kao i funkcionalne i imperativne stilove programiranja (uključujući objktno-orijentirane i prototipne). [4]

2.5. React

React (također poznat kao React.js ili ReactJS) razvijen je od strane Facebook društvene mreže koja ga također održava i koristi za vlastite potrebe. Facebook je odlučio omogućiti besplatno korištenje Reacta što je utjecalo na popularnost ovog alata. Često nazivan frameworkom, React to ipak nije. Naime, React je JavaScript biblioteka kojoj je glavna primjena kreiranje komponenti korisničkih sučelja, a za izradu kompleksnih React aplikacija potrebno je koristiti dodatne alate ili biblioteke u kombinaciji s Reactom.

React aplikacije, kao što je i aplikacija za praćenje osobne potrošnje, kreirane su odnosno sastoje se od više ponovno upotrebljivih komponenti. Prilikom kreiranja React komponenti treba voditi računa o mogućnosti ponovne upotrebe pojedine komponente, a isto je moguće postići na način da svaku pojedinu komponentu zadužimo za samo određeni dio funkcionalnosti aplikacije koju izrađujemo.

React komponente mogu se definirati na dva načina odnosno kao funkcije ili kao klase. Slika 2.4. prikazuje React komponentu definiranu kao funkciju. Bitno je napomenuti kako ovako definirana React komponenta, za razliku od komponente definirane kao klasa, nema pristup *state* objektu te ostalim metodama.

```
const ExpenseDashboardPage = () => (
  <div>
    <ExpensesSummary />
    <ExpenseListFilters />
    <ExpenseList />
  </div>
);
```

Sl. 2.4. React komponenta definirana kao funkcija

Slikom 2.5. prikazana je React komponenta definirana kao klasa, a nastavno na sliku dano je pojašnjene iste.

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}
```

Sl. 2.5. React komponenta definirana kao klasa [5]

React komponente definirane kao klase imaju mogućnost korištenja *state* objekta. Ukoliko dođe do promjene spomenutog objekta, pozivaju se odgovarajuće metode koje utječu na prikaz odnosno manipulaciju elemenata.

Prilikom izrade React komponenti uobičajeno je koristiti JSX – *JavaScript Syntax Extension* koji je također kreiran od strane Facebooka. JSX predstavlja JS funkciju, a primjer JSX-a možemo vidjeti na gore prikazanim primjerima React komponenti.

2.6. Firebase

Firebase je platforma za razvijanje mobilnih i web aplikacija razvijena od strane Firebase Inc., a u 2014. kupljena je od strane Google-a. Firebase je *"Backend-as-a-Service – BaaS"* te dozvoljava developerima veći fokus na razvoj korisničkog dijela aplikacije budući da korištenjem Firebase-a nije potrebno kreirati i održavati pozadinsku stranu aplikacije odnosno "*backend*".

Firebase platforma za razvijanje aplikacija korisniku omogućava potpuno rješenje prilikom

kreiranja aplikacija, a pri tome ne zahtjeva kreiranje vlastitih servera, aplikacijsko programskog sučelja – API te baze podataka.

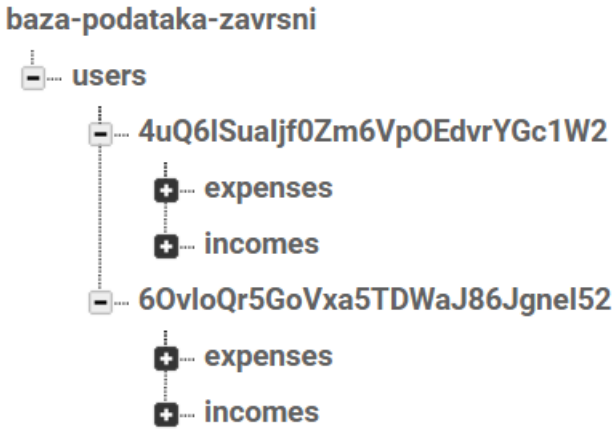
Platforma pruža mnoštvo servisa koji se koriste na velikom broju aplikacija, a za izradu Web aplikacije za praćenje osobne potrošnje najvažniji servisi svakako su:

- *Realtime data base* – pomoću ovog servisa omogućeno je spremanje prihoda i troškova u stvarnom vremenu
- *Firebase Auth* – omogućava autentifikaciju korisnika te podržava i prijavu odnosno autentifikaciju putem društvenih mreža kao što su Facebook, Google, Twitter i GitHub

Slijedećim potpoglavljem dostavljeno je detaljnije pojašnjenje Firebase Realtime baze podataka budući je ista od velike važnosti za izradu aplikacije.

2.6.1. Firebase Realtime baza podataka

Firebase Realtime database omogućava pohranu i sinkronizaciju podataka u stvarnom vremenu. Uneseni podaci pohranjeni su u oblaku u JSON formatu, a isti su također sinkronizirani u stvarnom vremenu za sve korisnike koji koriste pojedinu bazu podataka. Korištenjem Firebase *Realtime databasea* korisnicima je omogućena pohrana podataka u stvarnom vremenu te zaprimanje informacija o eventualnim ažuriranjima podataka također u stvarnom vremenu. Temeljem navedenog, bazu podataka izrađenu koristeći Firebase *Realtime database* možemo promatrati kao JSON stablo pohranjeno u oblaku, a koje se dodavanjem novih podataka dodatno grana. Slika 2.6. daje prikaz na podatke pohranjene u bazu podataka te je na istoj vidljivo grananje podataka.



Sl. 2.6. Prikaz podataka pohranjenih u Firebase bazu podataka

Također, bitno je napomenuti kako Firebase baza podataka omogućava kreiranje niza pravila kojima je moguće ograničiti pristup određenim podacima pohranjenima u bazu podataka. Definiranjem niza pravila korisniku je moguće onemogućiti pristup određenim "čvorovima" baze podataka. Navedena pravila od velike su važnosti prilikom osiguravanja kontroliranog pristupa bazi podataka, a ista se korištena i prilikom izrade Web aplikacije za praćenje osobne potrošnje.

2.7. Redux

Kako bi se olakšalo upravljanje stanjem odnosno "*state-om*" aplikacije korišten je Redux. Redux je predvidljivi spremnik stanja za JavaScript aplikacije [6]. Primjena ovog alata pomaže nam prilikom kreiranja aplikacije koje se uvijek ponašaju jednako te ih je lako testirati.

Redux u React aplikacijama omogućava kreiranje objekta "*store*" u kojem je pohranjeno globalno stanje aplikacije. Komponente aplikacije zatim mogu komunicirati sa "*store*" objektom te nije potrebno proslijeđivati "*state*" između komponenti.

Kako bismo promijenili stanje aplikacije pohranjeno u "*store*" objektu, potrebno je kreirati odgovarajuće akcije (engl. *actions*) te *reducere*. Akcije su JavaScript objekti koji su zaduženi za slanje informacija iz aplikacije prema "*store*" objektu, a moraju sadržavati tip (engl. *type*) te informaciju koju prenose. *Reducer* je čista funkcija (engl. *pure function*) koja kao parametre uzima trenutno stanje aplikacije i određenu akciju te vraća novo stanje aplikacije. Korištenjem ovog načina upravljanja stanjem aplikacije olakšalo je izradu iste.

3. WEB APLIKACIJA ZA PRAĆENJE OSOBNE POTROŠNJE

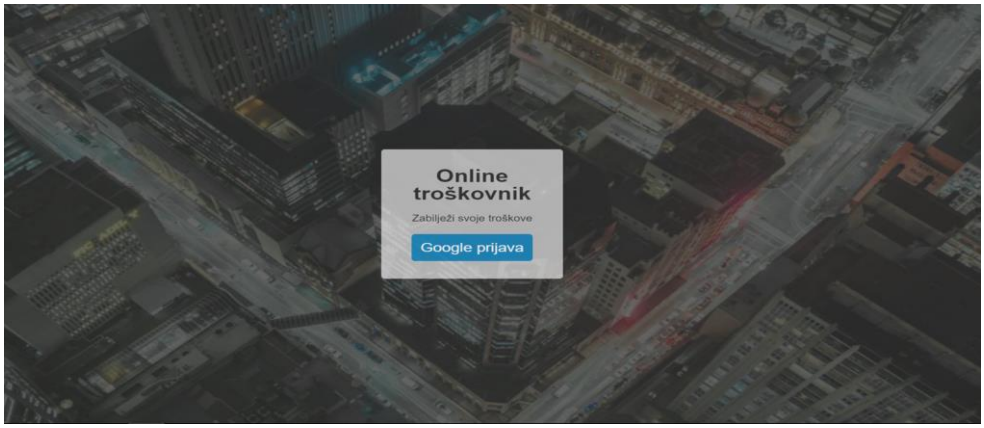
Glavna zadaća aplikacije kreirane prilikom izrade ovog završnog rada olakšati je krajnjem korisniku praćenje svakodnevne potrošnje te ukazati na eventualne prekomjerne troškove. Web aplikacija za praćenje osobne potrošnje korisniku putem privatnog korisničkog računa mora omogućiti jednostavan unos prihoda na mjesečnoj bazi te unos troškova na dnevnoj bazi. Također, korisniku je omogućeno naknadno uređivanje postojećih prihoda i troškova ili potpuno uklanjanje istih ukoliko se za tim ukaže potreba. Osim navedenog, korisnik ima mogućnost filtrirati unesene troškove po iznosu, datumu unosa troška ili samom nazivu troška, a u svrhu lakšeg praćenja istih.

Nastavno na prethodno poglavlje u kojemu su detaljnije opisane tehnologije primijenjene u izradi Web aplikacije za praćenje osobne potrošnje, ovim poglavljem prikazani su neki od važnijih dijelova koda na kojem se temelji aplikacija te je uz kod pruženo detaljnije pojašnjenje odnosno upute za lakše korištenje aplikacije.

Kako se svaka React aplikacija sastoji od većeg broja komponenti od kojih svaka obavlja točno određenu funkciju, nastavkom ovog poglavlja pružen je osvrt na glavne komponente Web aplikacije za praćenje osobne potrošnje odnosno kod na kojem se iste zasnivaju te na koji način utječu na funkcionalonst aplikacije.

3.1. Početna stranica i prijava na korisnički račun

Prilikom pokretanja aplikacije korisniku se na na zaslonu prikazuje početna stranica prikazana slikom 3.1. koja je detaljnije pojašnjena u nastavku.



Sl. 3.1. Početna stranica

Klikom na plavi gumb "Google prijava" otvara se novi prozor u kojem je potrebno izvršiti prijavu na postojeći Google korisnički račun kako bi se omogućilo daljnje korištenje aplikacije.

Navedenim načinom osigurano je da svaki korisnik ima pristup isključivo vlastitim prihodima i troškovaima odnosno vlastitim korisničkim podacima. Prilikom prijave novog korisnika odnosno prijave putem novog Google računa u bazi podataka kreira se novo "stablo" u JSON formatu sa posebnom ID vrijednošću za svakog korisnika te je korisniku omogućen pristup podacima isključivo iz vlastitog čvora. Autentifikacija odnosno prijava na korisnički račun omogućena je putem postojećeg Google računa. Unosom ispravnih korisničkih podataka za postojeći Google račun korisniku je omogućeno daljenje korištenje aplikacije.

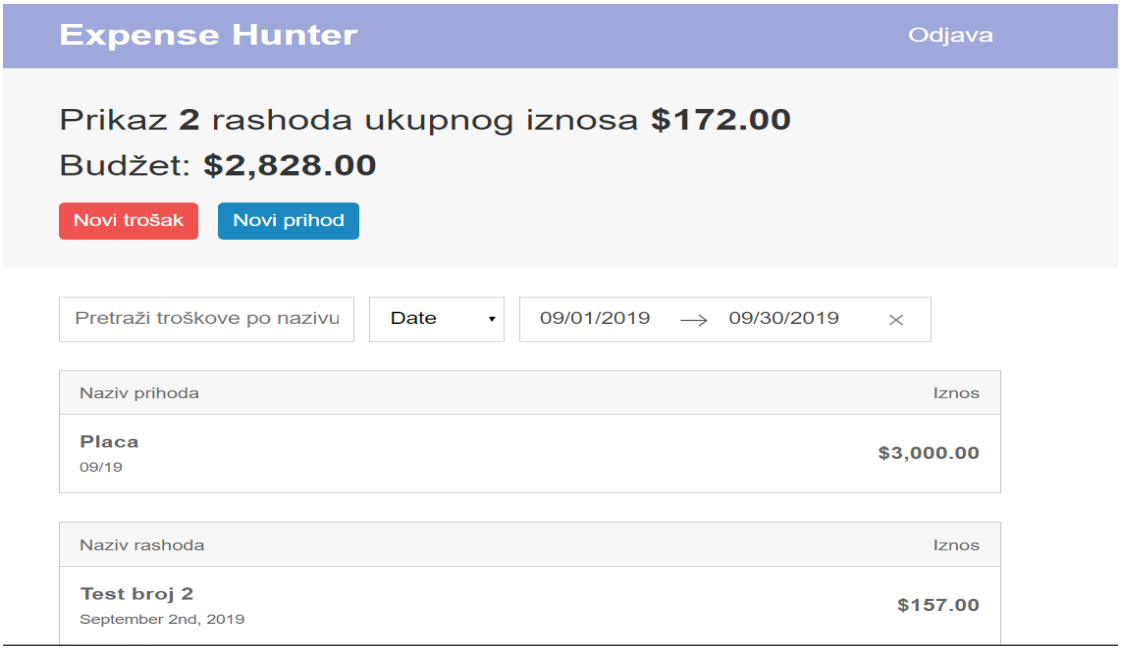
```
import { firebase, googleAuthProvider } from '../firebase/firebase';

export const login = (uid) => ({
  type: 'LOGIN',
  uid
});

export const startLogin = () => {
  return () => {
    return firebase.auth().signInWithPopup(googleAuthProvider);
  };
};
```

Sl. 3.2. Akcija za prijavu putem Google autentifikacije

Slikom 3.2. prikazana je funkcija "*startLogin*" koja koristeći metodu auth(), a pomoću Firebase parametra *googleAuthProvider* omogućava prijavu u aplikaciju putem Google korisničkog računa. Parametar *googleAuthProvider* prije upotrebe potrebno je uvesti odnosno "*importati*" u konfiguracijsku datoteku za Firebase.

3.2. Prikaz prihoda i troškova

Nakon uspješne prijave putem Google računa, korisniku je omogućen unos prihoda i ttroškova, te pregled i uređivanje eventualno prethodno unesenih prihoda i troškova za pojedino razdoblje. Kao što je prethodno navedeno, svaki korisnik ima uvid isključivo u svoje korisničke podatke. Slikom 3.3. prikazana je glavna stranica aplikacije na kojoj se nalaze svi uneseni troškovi te prihodi za odabrano razdoblje sortirani ovisno o odabranim filtrima.

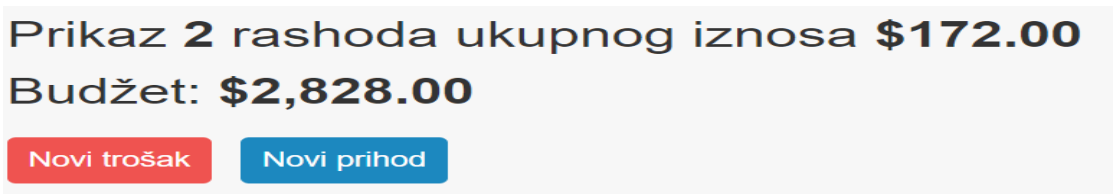


Sl. 3.3. Prikaz unesenih prihoda i troškova

Slikom 3.3. prikazan je izgled React komponente koja služi za prikaz svih unesenih prihoda i troškova, a na kojoj je moguće birati razdoblje za koje želimo pregledati unesene prihode i troškove te način na koji želimo sortirati iste. Kod ove komponente funkcija je koja služi isključivo za prikaz drugih React komponenti od kojih svaka ima određenu funkciju, a od koje se sastoji gore prikazana stranica aplikacije. U nastavku je detaljnije pojašnjenje komponenti koje služe za prikaz prihoda i troškova te sortiranje istih, a koje se prikazuju na gore prikazanoj stranici aplikacije.

3.3. Prikaz prihoda i ukupnog iznosa unesenih troškova

Jedna od komponenti koje se prikazuju odnosno "*renderaju*" na stranici prikazanoj slikom 3.3. je komponenta za prikaz budžeta i ukupnog iznosa troškova za odabrano razdoblje. Ista je prikazana slikom 3.4. niže.



Sl. 3.4. Prikaz budžeta i ukupnog iznosa troškova ovisno o postavljenim filtrima

Funkcija komponente prikazane slikom 3.4. je prikazati preostali iznos raspoloživih novčanih sredstava u odnosu na unesene troškove koji su vidljivi odnosno koji su obuhvaćeni postavljenim

parametrima filtriranja. Ukoliko se troškovi filtriraju po datumu unosa te se obuhvati cijelo razdoblje određenog mjeseca, iznos budžeta predstavljat će razliku unesenog prihoda te troškova ostvarenih u tom mjesecu. Međutim, ukoliko primjerice troškove filtriramo prema nazivu, iznos budžeta predstavjat će razliku prihoda unesenog za odabrano razdoblje i unesenog troška, tj. troškova obuhvaćenih filtriranjem prema nazivu. Također, prikazana komponenta sadrži dva gumba za dodavanje prihoda i troškova koji vode na novu komponentu zaduženu za spomenuto, a bit će dodatno pojašnjena u nastavku. Kod komponente prikazan je slikama u nastavku.

```
import React from 'react';
import { connect } from 'react-redux';
import { Link } from 'react-router-dom';
import numeral from 'numeral';
import selectExpenses from '../selectors/expenses';
import selectExpensesTotal from '../selectors/expenses-total';
import { findIncome } from './Income'

export const ExpensesSummary = ({ expenseCount, expensesTotal, income = {} }) => {
  const expenseWord = expenseCount === 1 ? 'rashod' : 'rashoda'
  const formattedExpensesTotal = numeral(expensesTotal / 100).format('$0,0.00')
  const { amount = 0 } = income
  const formattedBudget = numeral((amount - expensesTotal) / 100).format('$0,0.00')

  return (
    <div className="page-header">
      <div className="content-wrapper">
        <h1 className="page-header__title">Prikaz <span>{expenseCount}</span> {expenseWord} ukupnog iznosa <span>{formattedExpensesTotal}</span></h1>
        <h1 className="page-header__title">Budžet: <span>{formattedBudget}</span></h1>
        <div className="page-header__actions">
          <Link className="button" to="/create">Novi trošak</Link>
          <Link className="button" to="/createInc">Novi prihod</Link>
        </div>
      </div>
    </div>
  );
};
```

Sl. 3.5. Kod komponente zadužene za prikaz vidljivih troškova i raspoloživog budžeta

Slikom 3.5. prikazan je kod pomoću kojeg se prikazuju vidljivi troškovi ovisno o postavljenim parametrima filtriranja te razlika raspoloživog budžeta i prikazanih troškova.

Slikom 3.6. prikazan je dio koda gore spomenute komponente koji preuzima tj. "*mapira*" potrebne podatke iz "*store*" objekta te ovisno o određenoj promjeni, u ovom slučaju odabranih parametara filtriranja, uzrokuje promjenu stanja aplikacije.

```
const mapStateToProps = (state) => {
  const visibleExpenses = selectExpenses(state.expenses, state.filters);

  return {
    expenseCount: visibleExpenses.length,
    expensesTotal: selectExpensesTotal(visibleExpenses),
    income: findIncome(state.incomes, state.filters),
    filters: state.filters,
    incomes: state.incomes,
  };
};

export default connect(mapStateToProps)(ExpensesSummary);
```

Sl 3.6. Mapiranje globalnog stanja u komponentu

3.4. Filtriranje i sortiranje

Filtriranje troškova od velike je važnosti za funkcionalnost aplikacije te aplikaciju čini preglednijom i lakšom za korištenje. Kao što je prethodno navedeno, troškove je moguće filtrirati ovisno o datumu unosa i nazivu te sortirati ovisno o iznosu troška ili datumu. Bez mogućnosti filtriranja, popis troškova bio bi znatno nepregledniji te bi snalaženje u samoj aplikaciji bilo otežano. Slikom 3.7. pružen je prikaz komponente zadužene za filtriranje troškova.

Sl 3.7. Komponenta za filtriranje unesenih troškova

Unosom naziva troška u polje za unos teksta omogućeno je sortiranje po nazivu dok je klikom na padajući izbornik moguće odabrati želimo li troškove sortirati po datumu unosa ili iznosu. Ukoliko se odabere sortiranje po iznosu, trošak najvećeg iznosa bit će prvi na popisu dok u slučaju sortiranja po datumu prvo mjesto na popisu troškova zauzima trošak s najnovijim datumom unosa. Klikom na datum od kojeg odnosno do kojeg se prikazuju troškovi otvara se kalendar putem kojeg je omogućena promjena istih.

Kako se komponenta zadužena za filtriranje troškova sastoji od nešto duljeg koda, slikama niže prikazani su samo najvažniji dijelovi istog. Nastavno na slike koje prikazuju kod pruženo je pojašnjenje istog.

```
export class ExpenseListFilters extends React.Component {
  state = {
    calendarFocused: null
  };
  onDatesChange = ({ startDate, endDate }) => {
    this.props.setStartDate(startDate);
    this.props.setEndDate(endDate);
  };
  onFocusChange = (calendarFocused) => {
    this.setState(() => ({ calendarFocused }));
  }
  onTextChange = (e) => {
    this.props.setTextFilter(e.target.value);
  };
  onSortChange = (e) => {
    if (e.target.value === 'date') {
      this.props.sortByDate();
    } else if (e.target.value === 'amount') {
      this.props.sortByAmount();
    }
  };
```

Sl 3.8. Funkcije za promjenu parametara filtriranja troškova

Slikom 3.8. prikazane su funkcije pomoću kojih je omogućena promjena parametara filtriranja troškova, ovisno o korisničkom unosu odnosno "*inputu*". Prikazane funkcije pridružene su odgovarajućim JSX elementima putem kojih je korisniku omogućena interakcija s aplikacijom te promjena načina filtriranja troškova.

Funkcija *"onDatesChange"* zadužena je za promjenu početnog i krajnjeg datuma prikaza troškova, a u istoj se pozivaju akcije *"setStartDate*" i *"setEndDate"* pomoću kojih se mijenja stanje aplikacije. Funkcijom *"onFocusChange"* omogućen je prikaz kalendara nakon što korisnik klikne na JSX element za prikaz početnog i krajnjeg datuma za koji su prikazani troškovi. Filtriranje troškova po nazivu omogućeno je korištenjem funkcije *"onTextChange"* u kojoj se kao i u funkciji "*onDatesChange"* također poziva akcija definirana u posebnoj JavaScript datoteci. Odabir filtriranja po iznosu ili datumu unosa troška omogućen je pomoću funkcije *"onSortChange"*, a u kojoj se ovisno o korisnikovom odabiru izvršava određeni dio funkcije. Ukoliko korisnik odabere sortiranje po datumu izvršit će se dio funkcije u kojem se poziva akcija *"sortByDate"* dok se u slučaju odabira sortiranja po iznosu izvršava dio funkcije u kojem se poziva akcija *"sortByAmount"*. Kao što je prethodno navedeno, akcije koje se pozivaju u spomenutim funkcijama definirane su u posebnoj datoteci, a iste su zadužene za prenošenje potrebnih

informacija prema odgovarajućem *reduceru* koji zatim mijenja stanje aplikacije.

Omogućavanje pristupa globalnom stanju aplikacije, čime je ujedno omogućena i funkcionalnost gore spomenutih funkcija s obzirom da iste u protivnom ne bi uzrokovale promjenu stanja, realizirano je dijelom koda vidljivim na slici 3.9. koja je prikazana niže.

```
const mapStateToProps = (state) => ({
  filters: state.filters
});

const mapDispatchToProps = (dispatch) => ({
  setTextFilter: (text) => dispatch(setTextFilter(text)),
  sortByDate: () => dispatch(sortByDate()),
  sortByAmount: () => dispatch(sortByAmount()),
  setStartDate: (startDate) => dispatch(setStartDate(startDate)),
  setEndDate: (endDate) => dispatch(setEndDate(endDate))
});

export default connect(mapStateToProps, mapDispatchToProps)(ExpenseListFilters);
```

Sl. 3.9. Povezivanje komponente za filtriranje na "store" objekt

Slikom 3.9. prikazan je dio koda pomoću kojeg se komponenta za filtriranje troškova povezuje na "*store*" objekt zadužen za pohranu globalnog stanja aplikacije. Metodom *"mapDispatchToProps"* omogućeno je proslijeđivanje podataka komponente prema "*store*" objektu u kojem je pohranjeno globalno stanje aplikacije. Kao što je vidljivo na slici, spomenutom metodom vrši se "*dispatch*" prema "*store*" objektu koji uzrokuje ponovni *render* aplikacije u ovisnosti o korisnikovom unosu.

3.5. Unos prihoda i troškova

Kao što je navedeno u teorijskom dijelu završnog rada, prilikom kreiranja React komponenti potrebno je voditi računa o mogućnosti ponovne upotrebe komponente. Navedeno je moguće ostvariti na način da svaka komponenta obavlja samo određeni dio funkcionalnosti aplikacije. Svojstvo ponovne upotrebe React komponenata u izradi aplikacije najbolje je prikazano i primijenjeno prilikom kreiranja komponente za dodavanja prihoda odnosno troškova. Naime, klikom na gumb "Novi trošak" ili "Novi prihod" koji su jasno vidljivi na slici 3.4. otvara se ista React komponenta. Slikom 3.10. prikazan je kod koji služi za prikaz komponente za dodavanje prihoda i troškova.

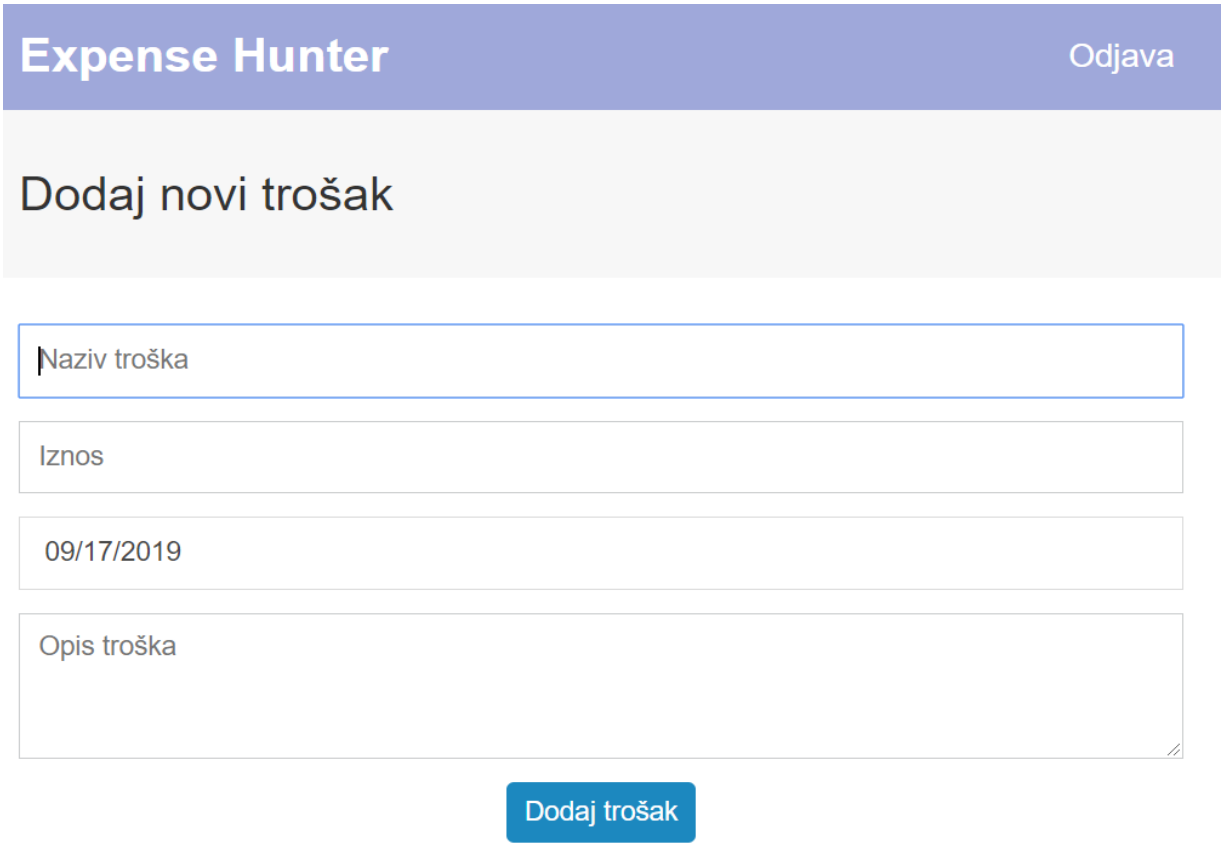
```
export class AddExpensePage extends React.Component {
  onSubmit = (submitData) => {
    const isExpense = location.pathname === '/create'

    if (isExpense) {
      this.props.startAddExpense(submitData);
    } else {
      this.props.startAddIncome(submitData)
    }

    this.props.history.push('/');
  };
  render() {
    const { location } = this.props
    const isExpense = location.pathname === '/create'

    return (
      <div >
        <div className="page-header">
          <div className="content-wrapper">
            <h1 className="page-header__title">{`Dodaj novi ${isExpense ? 'trošak' : 'prihod'}`}</h1>
          </div>
        </div>
        <div className="content-wrapper">
          <ExpenseForm
            onSubmit={this.onSubmit}
            isExpense={isExpense}
            />
        </div>
      </div>
    );
  }
}
```

Sl 3.10. Kod komponente za dodavanje prihoda i troškova

Prvim dijelom koda vidljivog na slici 3.10. gore prikazana je funkcija *"onSubmit"* koja se kao "*prop*" proslijeđuje u komponentu *ExpenseForm* koja je vidljiva u nastavku koda, a ista služi za slanje podataka prema bazi podataka nakon što korisnik klikne gumb za dodavanje troška ili prihoda. Prikazanim *if* grananjem (engl. *if statement*) provjerava se radi li se o dodavanju prihoda ili troškova te se ovisno o rezultatu provjere izvršava se potrebna akcija. Ukoliko se dodaje trošak, izvršava se akcija "*startAddExpense"* dok se u drugom slučaju, tj. prilikom dodavanja prihoda izvršava akcija "*startAddIncome"*.

Bitno je za napomenuti kako je korisniku za određeno razdoblje, tj. mjesec omogućen unos samo jednog prihoda. Uneseni prihod za određeno razdoblje ujedno predstavlja i raspoloživi budžet. Uneseni prihod moguće je naknadno uređivati ovisno o potrebi ukoliko se ostvare dodatna primanja. Iako korisnik nema mogućnost unositi više različitih prihoda za pojedino razdoblje, prilikom dodavanja prihoda ili uređivanja istog omogućen je unos dodatnog opisa prihoda čime je osigurano lakše praćenje promjena iznosa i izvora prihoda.

Nastavno na navedeno te ovisno odabere li korisnik unos novog prihoda ili troška, otvara se odgovarajuća verzija komponente *ExpenseForm*. Slikom 3.11. prikazan je izgled komponente *ExpenseForm* prilikom unosa troška. Važno je napomenuti kako je jedina razlika prilikom unosa prihoda u odnosu na unos troškova mogućnost odabira datuma unosa. Naime, prilikom unosa troška možemo birati točan datum unosa dok je prilikom unosa prihoda omogućen odabir isključivo mjeseca za koji isti unosimo.



Sl. 3.11. Obrazac za unos troškova

Korisnik nakon popunjavanja obrasca prikazanog slikom 3.11. klikom na gumb "Dodaj trošak" kreira novi trošak te se vraća na početnu stranicu aplikacije prikazanu slikom 3.3.. Ukoliko korisnik pokuša kliknuti na gumb za dodavanje troška, bez prethodnog unosa naziva i iznosa troška, na ekranu će se prikazati upozorenje te će se od korisnika tražiti unos navedenih podataka bez kojih dodavanje troška nije moguće. Povratak na glavnu stranicu aplikacije omogućen je klikom na naziv aplikacije u gornjem lijevom kutu ekrana.

Nastavno na prethodno spomenutu razliku koja se javlja prilikom unosa prihoda i troškova, slikom 3.12. prikazan je dio koda zadužen za promjenu mogućnosti odabira datuma unosa ovisno o tome radi li se o unosu troška ili prihoda.

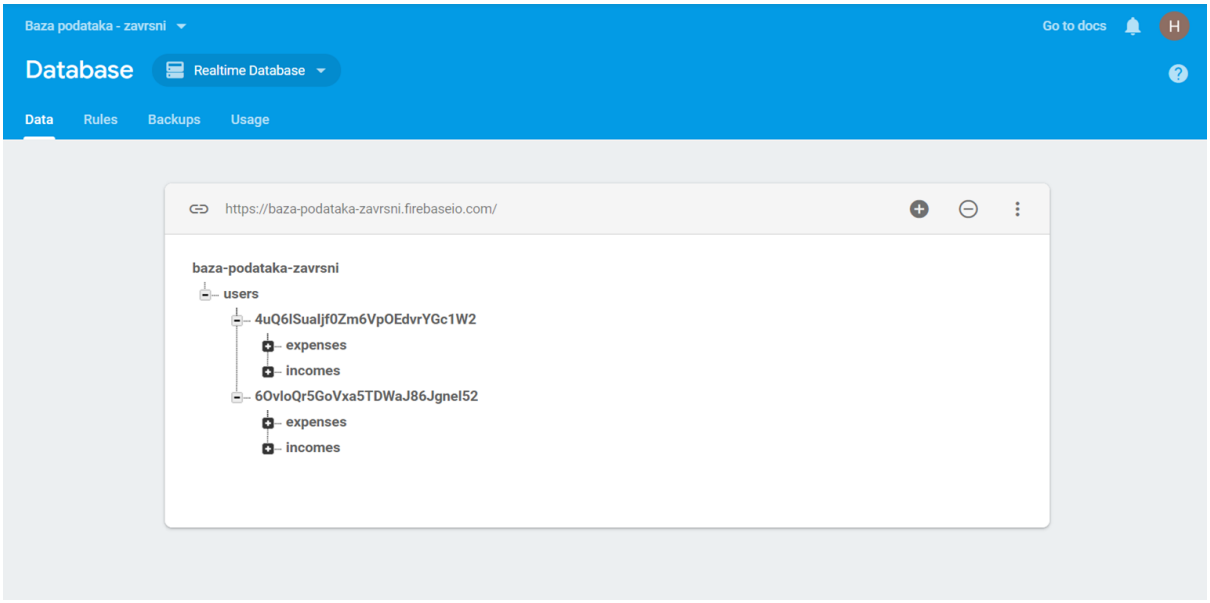
```
isExpense
? (
    <SingleDatePicker
      date={this.state.createdAt}
      onDateChange={this.onDateChange}
      focused={this.state.calendarFocused}
      onFocusChange={this.onFocusChange}
      numberOfMonths={1}
      isOutsideRange={() => false}
    />
) : (
  <MonthPickerInput year={!income ? moment().year() : moment(income.createdAt).year()}
    className="text-input"
    month={!income ? moment().month() : moment(income.createdAt).month()}
    onChange={(maskedValue, selectedYear, selectedMonth) => {
      getIncomeByMaskedValue(maskedValue, uid).then(income => {
        let nextState

        if (!income) {
          nextState = {
            createdAt: maskedValue,
            description: '',
            note: '',
            amount: '',
            calendarFocused: false,
            error: ''
          }
        } else {
          nextState = { ...income }
        }

        this.setState((prevState) => ({ ...prevState, ...nextState }))
      })
    }}
  />
```

Sl. 3.12. Odabir formata za unos datuma

Kodom prikazanim slikom 3.12. utvrđuje se radi li se o unosu troška ili prihoda. Ukoliko se radi o unosu troška, datum unosa odabire se putem komponente "*SingleDatePicker*" koja omogućuje odabir dana, mjeseca i godine. Međutim, prilikom unosa prihoda koristi se komponenta "*MonthPickerInput*" te ista omogućuje odabir mjeseca i godine dok odabir dana nije moguć. Kako je za pojedino razdoblje moguće dodati samo jedan prihod, prilikom pokretanja komponente za dodavanje prihoda utvrđuje se postoji li već unesen prihod za odabrano razdoblje ili ne. Ukoliko već postoji uneseni prihod, korisniku se omogućuje uređivanje postojećih podataka o unesenom prihodu dok se u protivnom otvara prazna forma za unos prihoda.

3.6. Firebase sučelje

Uneseni prihodi i troškovi spremaju se u bazu podataka izrađenu pomoću *Firebase Realtime Database*. Slikom 3.13. prikazana je Firebase baza podataka.



Sl. 3.13. Firebase sučelje i baza podataka

Kao što je vidljivo na slici 3.13. gore, prilikom prve prijave putem Google korisničkog računa za svakog korisnika kreira se jedinstveni ID. Također, vidljivo je kako svaki korisnik ima pristup samo podacima iz svog "čvora".

4. ZAKLJUČAK

Stil života današnjeg vremena u kojem je broj distrakcija sve veći, a ritam života sve brži, često može dovesti do zanemarivanja svakodnevnih detalja koji dugoročno mogu uzrokovati formiranja loših životnih navika. Upravo zbog navedenog kontrola ostvarenih troškova na dnevnoj bazi od iznimne je važnosti te manjak svijesti o eventualno prekomjernoj potrošnji može uzrokovati značajne probleme.

Izrađena aplikacija predstavlja jednostavno i lako primjenjivo rješenje za praćenje osobne potrošnje te korisnicima nudi mogućnost da u svega nekoliko trenutaka zabilježe ostvarene troškove i na taj način uspješno kontroliraju iste. Redovno praćenje ostvarenih troškova pruža nam mogućnost izbjegavanja prekomjerne potrošnje, ali nam nudi i povratnu informaciju o tome koja vrsta troškova zauzima najveći dio raspoloživog budžeta te na koji način možemo korigirati raspodjelu raspoloživim novčanim sredstvima ukoliko je isto potrebno.

Bolje razumijevanje osobnih navika te načina na koji funkcioniramo uvelike nam može pomoći te je nužno za ispravljanje i otklanjanje ponašanja koja nam dugoročno gledajući mogu naštetiti. Informacije o osobnoj potrošnji koje ostvarujemo korištenjem izrađene aplikacije od velike su pomoći te iste možemo koristiti kako bi izbjegli sticanje loših potrošačkih navika ili otklonili postojeće potrošačke navike koje nam ne idu u korist.

LITERATURA

[1] https://didu2017.wordpress.com/2017/04/20/html/ (posjećeno 28. svibnja 2019.)

[2] https://www.w3schools.com/html/html_elements.asp (posjećeno 28. svibnja 2019.)

[3] https://www.w3schools.com/css/css_syntax.asp (posjećeno 31. svibnja 2019.)

[4] https://en.wikipedia.org/wiki/JavaScript (posjećeno 1. lipnja 2019.)

[5] https://reactjs.org/ (posjećeno 2. lipnja 2019.)

[6] https://redux.js.org/introduction/getting-started (posjećeno 9. srpnja 2019.)

SAŽETAK

Cilj ovog završnog rada bio je izraditi web aplikaciju za praćenje osobne potrošnje koja korisniku na jednostavan način omogućuje kontrolu vlastitog budžeta i raspodjelu raspoloživih novčanih sredstava. Teorijskim dijelom rada pružen je osvrt na korisnost aplikacije te su opisane tehnologije koje su korištene prilikom izrade aplikacije od kojih je React JS JavaScript biblioteka svakako najvažnija. Izrađena aplikacija putem privatnog korisničkog računa omogućuje unos mjesečnih prihoda te troškova na dnevnoj bazi kao i sortiranje unesenih troškova ovisno o nazivu troška, datumu unosa ili iznosu troška.

Ključne riječi: prihodi, troškovi, React JS, aplikacija, praćenje potrošnje

ABSTRACT

Web application for tracking personal expenses

Main goal of this thesis was to create a web application for tracking personal expenses that will enable the user to easily control personal budget and manage available money. Theoretical part of the thesis focuses on describing the advantages of using the application and providing more detailed description of all technologies used to create the application of which the most important technology is JavaScript library React JS. Application enables user to input monthly earnings and everyday expenses. User is able to sort expenses by date of entry, amount or name.

Key words: expenses, earnings, React JS, application, tracking expenses

ŽIVOTOPIS

Hrvoje Vučetić rođen je 10. kolovoza 1995. godine u Starim Mikanovcima, Hrvatska. Osnovnoškolsko te srednjoškolsko obrazovanje završava u Vinkovcima. Srednju školu "Tehnička škola Ruđera Boškovića Vinkovci" smjer "Tehničar za mehatroniku" završava 2014. godine. Nakon završetka srednjoškolskog obrazovanja te uspješnog polaganja državne mature, upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, stručni studij elektrotehnike, smjer informatika.